Määrittelydokumentti

Toteutetaan shakkitekoäly ja pelimoottori. Moottorin tulee osata kaikki shakin säännöt (mukaan lukien tornitus, en passant ja 3 toiston sääntö. Mahdollisuuksien mukaan myös 50 siirron tasapeli) ja kyetä ilmoittamaan onko pyydetty siirto oikea vai väärä. Tämän tehdäkseen tulee moottorin kyetä pitämään kirjaa pelilaudan tilanteesta kullakin hetkellä ja kommunikoimaan pelin tilannetta käyttöliittymälle. Kommunikaatio tullaan toteuttamaan XBoard ja Lichess ympäristöjen käyttöliittymiin hyödyntäen valmista pohjaa (https://github.com/TiraLabra/chess)

Moottori

Moottori tulee ylläpitämään laudan tilaa 128 paikkaisessa 1-ulotteisessa taulukossa. Tämä nk. 0x88 esitystapa nopeuttaa nappuloiden siirtojen generoimista mahdollistaen O(1) aikaisen tarkistuksen siitä onko nappula siirron jälkeen laudalla. Kun jokaiselle liikkuma suunnalle (ja ratsuille, joiden liikkuminen poikkeaa) määritetään numeraalinen arvo (esimerkiksi oikealle liikkuminen olisi 1 ja vasemmalle -1), voidaan nappulan nykyiseen sijaintiin laudalla lisätä annettu luku ja nähdään uusi sijainti ilman monimutkaisia algoritmeja. Valelauta oikean laudan oikealla puolella antaa mahdollisuuden käyttää 0x88 hexadesimaalin binäärimuotoa tarkistamaan onko nappula ylittänyt rivit oikealta. Tämä on mahdollista, koska yksikään oikean laudan luvuista ei ole jaollinen 8:lla tai 128:lla (eli jos “ruudun numero” + 1000 1000 == 0, on nappula laudalla, muussa tapauksessa ei). Muissa suunnissa voidaan siirron osuminen laudalle tarkistaa yksinkertaisella yhteenlaskulla, jonka tuloksen ollessa väliltä 0 - 127, siirto on laudalla. Lisäksi tulee huomioida mahdolliset tielle osuvat nappulat, tämä onnistuu esimerkiksi jakamalla siirron määränpää ruutua siirtosuunnan luvulla (aikavaatimus O(n)).

Lautaa tulee päivittää käyttöliittymässä tehdyn siirron jälkeen APIlta tulevan UCI merkkijonon pohjalta. UCI sisältää tiedot nappuloiden nykyisestä sijainnista, mahdollisesta en passant tilanteesta, tornitusoikeuksista ja tehdyistä siirroista. Tiedot parsitaan ja täydennetään taulukkoon (aikavaatimus O(n)).

Lisäksi moottorissa tulee tekoälyä varten olla toiminnallisuudet siirron tekemistä ja perumista varten ja UCIn luomiseen siirron tapahtuessa.

Tekoäly

Tekoälyn tärkein tehtävä on hakea paras mahdollinen siirto. Tämä toteutetaan binääripuuta hyödyntävää Minimax algoritmia käyttäen. Minimax algoritmi mahdollistaa riittävällä syvyydellä parhaan siirron varman löytämisen, käytännössä täytyy tarkkuuden ja laskentaan käytettävän ajan suhteen tehdä kompromisseja. Minimax algoritmi käy läpi kaikki tämän hetken siirrot ja niille alisiirrot laskien niille vahvuudet hyödyntäen määritettyjä lukuarvoja (esim. Nappuloiden arvot, ruutujen arvot yms.). Minimax algoritmin aikavaatimus onkin bruteforce luonteensa vuoksi valitettavan hidas (O(n*d), jossa n on siirtojen määrä ja d syvyys). Algoritmia voidaan optimoida Alpha-beta pruningiksi kutsutulla tekniikalla, jonka avulla voidaan hylätä siirto mikäli yksikään alisiirto antaa nykyistä tilannetta huonomman tilanteen. Tekoäly hyödyntää moottorin antamia tietoja sallittujen siirtojen luomiseksi.